The Veracious Counting Bloom Filter

Brindha Palanisamy[1] and Senthilkumar Athappan[2]
[1]Research Scholar, Anna University, Chennai, India
[2]Professor and Head, Electrical and Electronics Engineering, Dr. Mahalingam College of Engineering and Technology, India

Abstract: *Counting Bloom Filters (CBFs) are widely employed in many applications for fast membership queries. CBF works on dynamic sets rather than a static set via item insertions and deletions. CBF allows false positive, but not false negative. The B_h-Counting Bloom Filter (B_h-CBF) and Variable Increment Counting Bloom Filter (VI-CBF) are introduced to reduce the false positive probability, but they suffer from memory overhead and hardware complexity. In this paper, we proposed a multilevel optimization approach named as Veracious B_h-Counting Bloom Filter (VB_h-CBF) and Veracious Variable increment Counting Bloom Filter (VVI-CBF) by partitioning the counter vector into multiple levels to reduce the False Positive Probability (FPP) and to limit the memory requirement. The experiment result shows that the false positive probability and total memory size are reduced by 65.1%, 67.74% and 20.26%, 41.29% respectively compared to basic B_h-CBF and VI-CBF.*

Key words: *Bloom Filter, false positive, Counting Bloom Filter, Intrusion Detection system*

Received August 3, 2014; accepted November 25, 2015

1. Introduction

1.1. Motivation

A significant security problem for networked systems is hostile or at least unwanted trespass by users or software. User trespass can take the form of unauthorized logon to a machine or in the case of an authorized user, acquisition of privileges or performance of actions beyond those that have been authorized. Software trespass can take the form of virus, worm or Trojan horse. Intrusion Detection Systems (IDS) have been developed to provide early warning of an intrusion, so that defensive action can be taken to prevent or minimize damage.

Early developed IDS are mainly of two types: (1) Host based IDS (2) Network Based IDS. Again, all the intrusion detection systems use one of the two detection techniques: (a) Statistical anomaly (b) Signature based IDS. The software based solution cannot catch up with the gigabit network. This impediment leads to a demand of hardware solution to speed up the process.

1.2. Intuition for Veracious Variable CBF

Burton H. Bloom introduced a new hash coding method. This method is suggested for application in which the great majority of messages to be tested will not belong to the large set. Firstly, the average time required for classifying the element as a non-member of large set is high. Secondly, the probability of error should be minimized (i.e.) the false identification of the member to be in the set will create a small error [3, 4, 6]. Thirdly, computation time and space should be efficient to meet the practical applications.

Counting Bloom Filter emerged to overcome the pitfalls of BF. CBF have been contrived for multi-set representation that may be dynamic due to increment/decrement. This aspect makes it as a natural virtue in many networking solicitations [14, 20, 11, 12].

The B_h sequence is now used to improve the performance of CBF [16]. Basic CBF will increment the counter by 1, which is constant throughout the entries. While B_h-CBF have two counters. First counter will have a constant increment of one, it counts the number of elements hashed into the CBF. Second counter will have an inconstant incremental of weighted sum of the elements.

The B_h-CBF nearly doubles the counter used in the design. To overcome this limitation, Variable Increment Counting Bloom Filter (VI-CBF) emerged [19], which also uses a variable increment counter but it relies only on one counter per entry. It does not rely on the counter that counts the number of hashed elements into the CBF.

Upon this the VI-CBF also suffers from added complexity and memory overhead. To counter act this we proposed two new designs named Veracious B_h Counting Bloom Filter (VB_h-CBF) and Veracious Variable Increment Counting Bloom Filter (VVI-CBF). The goal is to reduce the false positive probability. The VB_h -BF and VVI-CBF is constructed by segmenting the counter vector into

multi levels. The counters are organized by offset indexing. In VB_h-CBF and VVI-CBF the first level is used to perform membership queries, while other levels are used for insertions and deletions. The simulation result shows that the VB_h -CBF and VVI-CBF outperforms B_h-CBF and VI-CBF in false positive probability at the same memory consumption.

The rest of this paper is organized as follows. Section II introduces the background and related work of BF. Section III enumerates the algorithms of B_h-CBF and VI-CBF. We described the construction of VB_h-CBF and VVI-CBF in Section IV. Finally, the experimental results are shown in Section V to validate the construction of VB_h-CBF and VVI-CBF. Section VI deals with inferences of the work discussed in the earlier sections.

2. Background and Related Works

A Bloom Filter (BF) is a compact space efficient data structure, which consists of an array of m-bits whose values are initially set to zero. It is used to represent a set $S=\{X_1,\ldots,X_n\}$ of n elements. For each element that is added to the set S, k different hash functions $h_1,\ldots..h_k$ with the range $\{1,\ldots..,m\}$ are engaged to manipulate k different hash values $h_1(X_i),\ldots\ldots,h_k(X_i)$. Based on these values that are mapped the bits $h_j(x_i)$ are set to 1 over j=1,2,…..,k. To check if a certain element Y is present in our BF, similar procedure is followed as used in insertion process [4][7][17][15][18]. If the bits $h_j(Y)$ equal to 1, then the element is probably in the set. If it is zero, then certainly it is not in the set. Sometimes BF suffers from false positive error and will not support deletion of an element. The test bits were set to 1 due to insertion of different elements. The false positive can be hacked by simply making trade-off between size of the BF and false positive probability.

A best way is provided to support deletion without recreating the filter anew named as Counting Bloom Filter (CBF) [10]. In CBF a single bit registers are replaced with n-bit counter. The insertion and deletion is established by simply incrementing and decrementing the counter [14][20][11][12]. As discussed in [20] the SRAM is not compatible with the high speed data matching. A new memory based on CAM introduced [1] to overcome the drawback of SRAM but still as network speed increases in Gbps, CBF are more preferred for intrusion detection. Still, CBF faces obstacles while the narrowness they impose on scalability and arithmetic overflow.

Further Mitzenmacher made a research to compress the BF to make it more optimal. By compressing the size of the bit array (z), the length of the uncompressed and compressed BF has found no gain. So the value of z is kept constant to minimize the FPP [17, 20]. The compressed BF also suffers from few limitations like memory overhead, computation cost and the size of the filter cannot be calculated since it is compressed and uncompressed every time.

A spectral BF is related to CBF, which is adapted for encoding multi-set. The filtering of elements is based on the multiplicities that are in specific range. The value of the multiplicity should satisfy $\widehat{fq} = fq$ otherwise error occurs. This error can be minimized by two ways proposed by Cohen and Matias as Minimal increase and Recurring minimum methods [9].

Chazelle *et al.,* proposed the Bloomier filter [8], which can associate values with the key stored in the BF and have one of the color values associated with each object. By querying the objects, the corresponding color is returned. Like BF, this structure also suffers from small FPP but not false negative.

Bonomi *et al.,* introduced an enhanced structure named d-left hashing which demands only half as much as space as CBF [5]. The above drawbacks of CBF are overcome by this d-left hashing.

Stable Bloom Filter (SBF) by Deng and Rafiei is designed to continuously expel the information to make space for the most recent elements. In this filter there is no way to store the entire data. Unlike BF, the SBF introduces false negative, since they expel the trite information. This technique works better than standard BD in terms of time efficiency and FPP when a small space and acceptable false positive rate is given [10].

Almeida *et al.,* introduced a variant of BF that can acclimate dynamically to the number of elements stored named as Scalable Bloom Filter. This method guaranteed for minimum FPP [2]. The scalable BF is built by arranging the BF sequentially with increasing capacity and tighter FPP, so as to ensure that a maximum false positive probability can be set beforehand, regardless of the number of elements to be inserted. In this paper, VBH-CBF and VVI-CBF are designed to improve the FPP by using multilevel optimization to avoid the previous limitations.

3. Concise Representation of Counting Bloom Filter, B_h-CBF and VI-CBF

3.1. Counting Bloom Filter

The CBF suggested by Fan et al. as discussed in section II is a generalized BF, in which the single bit bloom filter is replaced by a n-bit BF [11]. The insertion and deletion are done by simply incrementing and decrementing the counter. The low power Linear Feedback Shift Register (LFSR)

are employed for this design[8]. The operations are shown as Algorithm 1 steps as follows:

Algorithm 1: Insertion In CBF

```
Insert(Element x);
    for(i=1;i<=q;i++) do
        for(j=1;j<=k;j++)do
            p=hj(xi);
    Cj(p)++;
            end for;
        end for;
```

The element x is inserted into the BF using k independent hash functions. The corresponding bit array is incremented by 1, rest are retained as zero as shown in Algorithm 1. The reverse process is carried out to delete an element from the array.

Algorithm 2: Query Operation In CBF

```
Query(Element x);
for(i=1;i<=q;i++) do
        for(j=1;j<=k;j++)do
                p=hj(xi);
                if Cj(p)==0 then
                return FALSE;
                else
                return TRUE;
                end if;
        end for;
end for;
```

To query an element in the CBF the hash function is performed and the corresponding location in the bit array is checked for its presents [20]. If it equals 0 then definitely the element is not present, else it is present in the array as given by the Algorithm 2. The CBF also suffers from false positive error, so a new idea based on Sidon sequence is proposed [16][15].

3.2. SIDON Sequence

The Sidon Sequence is commonly known as *Bh*-set or *Bh* sequence [16]. We start with the basic Scenarios of B_h sequences. *B2* sequences are also called Sidon sequences.

- *Rule 1.* Let (X +) be an Abelian group. Let $G = \{U_1,U_2 \ldots U_n\}$ be a sequence of elements of *X*. Then *G* is a *Bh sequence* over *X* if all the sums $U_{i1}+U_{i2}+\ldots U_{in}$ with $1 \le i_1 \le i_2 \le \ldots \le i_n$ are distinct [11].
- *Example 1.* Let *G* = Z and $G = \{ U_1,U_2, U_3,U_4\} = \{1,4,11,13\} \subseteq G$.

We can see that all the 10 sums of 2 elements of *G* and20 sums of 3 elements are distinct. Therefore, *G* is a *B2* and *B3* sequence. However, the two sums of 4 elements, hence *G* is not a *B4* sequence.

3.3. Functionality of B_h Sequence

The B_h sequence is used to improve the performance of CBF [12, 10]. Basic CBF will increment the counter by 1, which is constant throughout the entries. While in B_h sequence based CBF there will be two counters [9].

First counter will have a constant increment of one, it counts the number of elements hashed into the CBF. Second counter will have an inconstant incremental of weighted sum of the elements as shown in Table.1.

There are three basic operations performed by each of these counters.

- Insertion or update phase.
- Deletion or removal phase.
- Query or test phase.

3.3.1. Element insertion

Algorithm 3, Shows the element 'A' is inserted whose G={1,4,11,13} and k=3. In this, $\{h_1(A),h_2(A),h_3(A)\} = \{1,4,8\}$ and $\{g_1(A),g_2(A),g_3(A)\}=\{2,1,4\}$. The element 'A' is hashed to 1,4,8 locations in the array using h_1, h_2 and h_3. It increment the counter C1 by one and it increments the second counter by hashing $\{U_{g1(A)}, U_{g2(A)}, U_{g3(A)}\} = \{U_2, U_1, U_4\} = \{4,1,13\}$ respectively.

Algorithm 3: Insertion Operation In B_h-CBF

```
Insert(Element A);
#To add the element in C1
for(i=1;i<=q;i++) do
    for(j=1;j<=4;j++)do
    p=hj(Ai);
    C1j(p)++;
# To add the element in C2
Get the value for U={1,4,11,13}
Ugj(A)=gj(A);
C2j(p)= C2j(p)+Ugj(A);
    end for;
end for;
```

3.3.2. Element querying

To query whether an element 'B' is present in the set S, both the counters are checked for its presence. If C1(*i*)=0, then the constant increment counter determines that 'AB' $\notin$ S.

The C2(*i*) is checked for exact sum of the a value based on G as shown in Algorithm 4.

3.3.3. Element deletion

The deletion of an element says 'AB' is performed similar to element insertion. First counter of $h_i(a)$ is decremented by one and second counter is decremented by $U_{gi}(a)$. The element 'AB' is removed by decrementing the first counter C1 using h_1 (AB), h_2 (AB) and h_3 (AB) by 1 and their second counter C2 by 13, 1 and 4 respectively [19].

Algorithm 4: Query Operation In B_h-CBF

Query(Element A)

#To query the element in C1
for(i=1;i<=q;i++) do
for(j=1;j<=4;j++)do
$p=h_j(A_i)$;
If C1j(p)==0 then
return FALSE;
else
return TRUE;
end if;
 # To query the element in C2
Get the value forU={1,4,11,13}
$U_{gj(A)}=g_j(A)$;
if C2j(P) contains the Value of D hashed by $g_j(A)$ then
return TRUE;
else
return FALSE;
end if;
end for;
end for;

3.4. Variable Increment Counting Bloom Filter

The B_h-CBF illustrated above has two counters which will certainly doubles the hardware complexity and memory utilization. The C1 and C2 counters are replaced with single counter that keeps track of the elements in the array. The B_h-CBF and VI-CBF shows 22.2% and 34% improvement in the total memory size. The hardware's complexity is reduced by a factor of two approximately. The Simulation results in the previous works [18] shows that the B_h-CBF and VI-CBF also suffers from small false positive rate.

4. Veracious Counting Bloom Filter

In this section, we present a multilevel optimization technique to B_h-CBF and VI-CBF and we name it as Veracious B_h-CBF and VI-CBF. We described the basic construction of VB_h-CBF and VI-CBF to reduce the FPP. Next, the proposed and standard techniques are compared to designate the improvement in the FPP.

4.1.VB_h-CBF Construction

The basic idea of VB_h-CBF is to partition the counter array into multilevel. Further, we segregate the insertion/deletion operations and query operations. Veracious B_h-CBF has a ranked structure which is composed of h-levels $b_1,\ldots,b_r$ and an idle array b_i. The VB_h-CBF uses k hash functions $h_1,\ldots,h_k$ to hash an element x into k bits in the first controlled by the parameter m for a given n and k [11]. In the ranked structure, b_1 is employed to substantiate membership query, $b_2,\ldots.b_r$ is used to manipulate the element insertion, b_i is utilized for further insertion of new elements. The first level b_1 has the same size as B_h-CBF. Since m is 10, the size of b_1=10. The remaining 21 bits are used for idle array b_i. The VB_h-CBF uses k hash functions $h_1,\ldots,h_k$ to hash an element x into k bits in the first level b_1 and l_1 is the bit size of b_1. The query operation is carried to check the array b_1 as shown in Algorithm 1.

Algorithm 5, Shows that if all bits are set to 1, then the element x is said to be in VB_h-CBF. We presume $|b_j|=l_{1j}$, where j is in the range [1,2…,r]. To traverse through the counter an offset index in b_j by using the function popcount(b_j,j). This function returns the number of ones counted while transverse through the position j. Fig. 1,shows the insertion of three elements x,y,z. The C1 is 10-bit array, position 8 in C1 and C2 are hashed by both y and z. So the first counter is incremented by one and the second counter is arranged in a ranked structure where its value traverses through two levels b_3 and b_4. As bit 8 in C1 is set to 1, the corresponding bit in b_2 is set to 1 based on the B_h concept. Next we call the popcount to index the position in b_2. It returns the value 3, so this helps to address the b_3. Similarly b_3 returns 1 which is used for b4. The sum of all the traverse path values gives the total counter value.

Algorithm 5: Query Operation In VB_h-CBF

Query(Element x),
#To query an element in C_1 and C_2
for(i=1;i<=k;i++)do
$q=h_i(x)$,
$p=U_{gi}(x)$ mod l_i;
If($b_1(p)$==0 and C1(q)==0)then
return FALSE;
end if;
end for;

In order to insert or to delete an element the corresponding counter is incremented or decremented. When an element is inserted in to VB_h-CBF, we need to traverse through the $b_1,\ldots..,b_r$ series. To insert an element we expand the next level b_{j+1}, by adding variable value from the set U indexed by the second hash function. Similarly, perform deletion by shrinking the last level b_j by shifting backward all the bits and decrementing b_j by the variable value from set U as in algorithm 6.

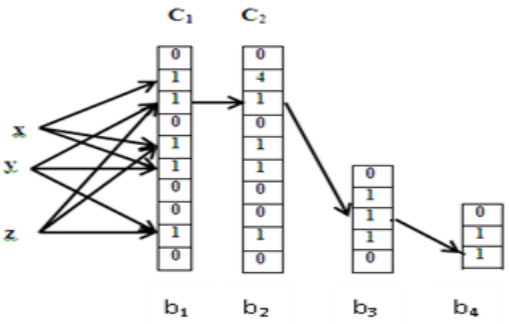

Figure 1. Vb_h-CBF with ranking structure

Algorithm 6: Insertion Operation In Vb_h-CBF

Insert(Element A)
#To insert the element in C1

for(i=1;i<=k;i++) do
q=h_i(A);
$C1_i$(p)++;
#To insert the element in C2
Get the value forU={1,4,11,13}
$U_{gj(x)}$=g_j(x);
for(j=1;j<=r;j++)do
if(j== $ugi(x)$)then
p=g_i(x)mod l_i;
end if;
if (b_1(p)=U_{gj}(x)) then
Offset=popcount(b_j,p);
P=offset;
#Expand b_j by one position b_{j+1}
Else
b_j(p)=U_{gj}(x);
offset=popcount(b_j,p);
expand(b_{j+1},offset);
p=offset;
b_{j+1}[p]=0;
exit();
end if; end for;

The VB_h-CBF, has two hash functions and their range varies from *{1,….,m}* and for second hash function *{1,2,…..l_1}*. The FPR for VB_h-CBF, is

$$FPR=\left(1-\sum_{j=0}^{h}\binom{nk}{j}\left(\frac{l_1-1}{l_2 m}\right)^{j}\left(1-\frac{1}{m}\right)^{nk-j}\right)^{k} \quad (1)$$

4.2. VVI-CBF Construction

The VB_h-CBF suggested above uses two counters per entry. This nearly doubles the hardware utilization. Moreover, the elements in the sum of C2 will not be unique when there are h elements. So we introduce VVI-CBF (Veracious Counting Bloom Filter) that uses only a single counter instead of two counters. As previously enumerated in Figure 1, we again use the array to store the elements. Like the second counter in VB_h-CBF, a single variable counter is updated based on the values selected in the set U. Upon insertion, the counter is incremented by $gi(x)$ in the set G positioned by $h_i(x)$. Similarly, deletion is accomplished by deleting the counter by $gi(x)$ in the set G positioned by $h_i(x)$. Another problem with VI-CBF scheme as discussed in previous works, it cannot directly use the bh scheme as there is no small counter to store the number of elements hashed into the counter[18][26].

To solve this [19] $D=D_L$ that does not need a small lookup table. We adopted this scheme to design VVI-CBF.

The FPR of the VI-CBF scheme is shown in eqn. (2),

$$FPR=\left(\begin{array}{l}1-\left(1-\frac{1}{m}\right)^{nk}-\frac{L-1}{L}\binom{nk}{1}\frac{1}{m}\left(1-\frac{1}{m}\right)^{nk-1}\\ -\frac{(L-1)(L+1)}{6L^2}\binom{nk}{2}\left(\frac{1}{m}\right)^{2}\left(1-\frac{1}{m}\right)^{nk-2}\end{array}\right)^{2} \quad (2)$$

The FPR of the VVI-CBF scheme is given by eqn. (3)

$$FPR=\left(\begin{array}{l}1-\left(1-\frac{1}{l_1}\right)^{nk}-\frac{L-1}{L}\binom{nk}{1}\frac{1}{l_1}\left(1-\frac{1}{l_1}\right)^{nk-1}\\ -\frac{(L-1)(L+1)}{6L^2}\binom{nk}{2}\left(\frac{1}{l_1}\right)^{nk-2}\end{array}\right)^{2} \quad (3)$$

Where L is the size of D_L, l_1 is the bit size of first level in b_j, n is the maximum number of elements and k is the number of hash functions. Definitely this will improve the false positive rate to a great extend for a CBF when it is chosen as m≥10.

5. Experiment Result

The experiment is conducted to analyze the performance of the VB_h-CBF and VVI-CBF. 50 signatures each of 8 bits per cycle are loaded in the database. The false positive rate is analyzed based on the test bench designed using Hardware Description Language (HDL). The simulation is conducted for 100 intrusion patterns. The functional blocks are implemented using Virtex4 (XC4VS25) Field Programmable Gate Array(FPGA).The false positive rate for B_h-CBF, VI-CBF, VB_h-CBF and VVI-CBF with $D=D_L$ are 0.00089, 0.00031, 0.00028 and 0.00010 respectively. The false positive rate improves by 65.4% and 67.74% respectively. As the number of entries increases the improvement would be seen over the order of magnitude.

The memory requirement comparison is shown in Table 1 where the previous works are shown on the left hand side and proposed two techniques are shown the right hand side. The performance results of the previous approaches are taken from [12][20]. We presented the five existing approaches and two new techniques VB_h-CBF and VVI-CBF.

We used the letter U instead of D in the previous works and their results are summarized in Table 1. The B_h-CBF and VB_h-CBF requires an addition table as explained in VVI-CBF Construction.

As the counter array is partitioned into Multi-levels the total memory requirement reduces and further, the segregation of insertion/deletion operations and query operations brings a better false positive rate. In VB_h-CBF, the memory size requires is 9.17 whereas in the VVI-CBF it drops to 6.13. A total memory size of 20.26% and 41.29% improvement are accomplished compared to B_h -CBF and VI-CBF.

Table 1. Memory Requirements comparison.

	Existing Techniques[8,10]					Proposed Techniques	
	CBF	Spectral BF	dl-CBF	B_h- CBF	VI-CBF $D=D_L$	VB_h-CBF	VVI-CBF $D=D_L$
Main Structure (KB)	14.1	8.12	5.2	12.07	10.97	9.17	6.13
Secondary Structures (KB)	-	-	-	-	-	0.44	0.31
Additional Tables(KB)	-	4	-	0.02	-	0.03	-
Total Size (KB)	14.1	12.12	5.2	12.09	10.97	9.64	6.44
Total Memory Size in %	-	-	-	-	-	20.26	41.29
False Positive Rate	10^{-3}	10^{-3}	$1.5x10^{-3}$	10^{-3}	10^{-3}	10^{-3}	10^{-3}
Computation Time(ns)	0.447	0.389	0.224	0.410	0.35	0.27	0.21

The computation time required for all the methods are addressed to show that there are improvement of 34.14% and 40% compared to B_h-CBF and VICBF. This improvement makes the design more significant for intrusion detection application where high speed detection exists.

6. Conclusion

The target of this paper is to propose an efficient technique to reduce the false positive rate and memory utilization with low computation time. We have also demonstrated that these two methods achieve a low false positive rate. Simulation result shows that the false positive probability is reduced by 65.4% and 67.74% compared to standard B_h-CBF and VI-CBF respectively. The proposed work suffers from little hardware overhead due to increase in B_h blocks. The proposed work is suited well for intrusion detection system since it accomplishes only low memory space.

References

[1] Ali Q., "A Flexible Design of Network Devices Using Reconfigurable Content Addressable memory," *The international Arab Journal of Information technology*, vol. 8, no.3, pp. 235-243, 2011.

[2] Almeida P., Baquero C., Preguica N., Hutchison D., "Scalable Bloom Filters," *Information Processing Letters*, vol. 101, no. 6, pp. 255-261, 2007.

[3] Beddin Gerav S., Ahmadi M., "Bloom filter applications in network security: A state-of-the-art survey," *Computer Networks*, vol. 57, no.18, pp. 4047-4064, 2013.

[4] Bloom B.H., "Space/time trade-offs in hash coding with allowable errors," *Comm. of the ACM*, vol. 13, no. 7, pp.422-426, 1970.

[5] Bonomi F., Mitzenmacher M., Panigrahy R., Singh S., Varghese G., "An Improved Construction for Counting Bloom Filters," *Proc. 14th conference on Annual European Symposium*, Springer-Verlag London, UK, pp. 684-695, 2006.

[6] Bose P., Guo H., Kranakis E., Maheshwari A., Morin P., Morrison J., Smid M., Tang Y., "On the false-positive rate of Bloom filters," *Information Processing Letters*, vol. 108, no.4, pp.210-213, 2008.

[7] Brindha P., SenthilKumar A., Mohanapriyaa V.P., "Survey and Evaluation of D Flipflop for Low Power Counter Design Using Sub-Micron Technology," *International Journal of Electronics Communication and Computer Engineering*, vol. 4, no. 2, pp. 339-343, 2012.

[8] Brindha P., SenthilKumar A., "Area Efficient Counting Bloom Filter (A-CBF) design for NIDS," *International Journal of Computer Applications*, vol. 70, no. 4, pp.17-21, 2013.

[9] Brindha P., SenthilKumar A., "Network Intrusion Detection System: An Improved Architecture to Reduce False Positive Rate," *Journal of Theoretical and Applied Information Technology*, vol. 66, no. 2, pp.618-626, 2014.

[10] Broder A., Mitzenmacher M., "Network Application of Bloom Filter: A Survey," *Internet Mathematics*, vol. 1, no. 4, pp. 484-509, 2004.

[11] Chazelle B., Kilian J., Rubinfeld R., Tal A., "The bloomier filter: an efficient data structure for static support lookup tables," *Proc.15th Annual ACM-SIAM Symp. On Discrete Algorithms*, Philadelphia, PA, USA, pp. 30-39, 2004.

[12] Cohen S., Matias Y., "Spectral bloom filters," *In SIGMOD '03: Proc. of the 2003 ACM SIGMOD Int. conference on Management of data*, New York, NY, USA, pp.241-252, 2003.

[13] Fan D., Davood R., "Approximately detecting duplicates for streaming data using stable bloom filters," *Proceedings of the ACM SIGMOD Conference*, New York, NY, USA, pp.25-36, 2006.

[14] Fan L., Cao P., Almeida J., Broder A., "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," *IEEE/ACM Trans. on Networking*, vol. 8, no.3, pp.281-293, 2000.

[15] Ficara D., DI Pietro A., Giordano S., Procissi G., Vitucci F., "Enhancing counting bloom filters through Huffman-coded multilayer structures," *IEEE/ACM Trans. on Networking*, vol. 18, no. 6, pp. 1977-1987, 2010.

[16] Graham S. W., "B_h sequences," P*rogress in mathematics*, vol. 138, pp.431-449, 1996.

[17] Guo D., Liu Y., YangLi X., Yang P., "False Negative Problem of Counting Bloom Filter," *IEEE Trans. on Knowledge and Data Engineering*, vol. 22. no. 5, pp. 651-664, 2010.

[18] Kun Huang, Jie Zhang, Dafang Zhang, Gaogang Xie, Salamatian K., Liu A.X., Wei Li, "A Multi-partitioning Approach to Building Fast and Accurate Counting Bloom Filters," *Proc. 27th International Symposium on Parallel & Distributed Processing (IPDPS)*, Boston, pp.1159-1170, 2013.

[19] Laufer R.P., Velloso P.B., Duarte O.C.M.B., "A Generalized Bloom Filter to Secure Distributed Network Applications," *Computer Networks*, vol. 55, no. 8, pp.1804-1819, 2011.

[20] Manjula G., Bridha P., "A Survey On Architectural Design Of Bloom Filter For Signature Detection," *International Journal of Engineering Research & Technology*, vol. 2, no.3, pp.1-6, 2013.

[21] Martin G., O'Bryant K., "Constructions of Generalized Sidon Sets," J*ournal of Combinatorial Theory, Series A*, vol. 113, no.4, pp.591-607, 2006.

[22] Mitzenmacher M, "Compressed bloom filters," *IEEE/ACM Trans. On Networking*, vol. 10, no. 5, pp. 604-612, 2002.

[23] Paynter M., Kocak T., "Fully Pipelined Bloom Filter Architecture," *IEEE Comm. Letter*, vol. 12, no. 11, pp. 855-857, 2008.

[24] Rottenstreich O., Keslassy I., "The Variable Increment Counting Bloom Filter," *INFOCOM, 2012 Proceedings IEEE*, Orlando, FL, pp.1880-1888, 2012.

[25] Safi E., Moshovos A., Veneris A., "L-CBF: A Low-Power, Fast Counting Bloom Filter Architecture," *IEEE Trans. on Very Large Scale Integration (VLSI) systems*, vol. 16, no.6, pp. 628-638, 2008.

[26] Wei Li, Kun Huang, Dafang Zhang, and Zheng Qin," Accurate Counting Bloom Filter For Large Scale Data Processing," *Mathematical Problems in Engineering*, pp. 1-11, 2013.

Brindha Palanisamy has received her B.E. Degree in Electronics and Communication Engineering Anna University, Chennai in 2006 and M.E. Degree in VLSI DESIGN from Anna University, Coimbatore in 2009. She is currently doing her research under Anna University, Chennai. Her current research is on Network Intrusion Detection System. Currently she is working as Assistant Professor (S. Gr.) at Velalar College of Engineering & Technology, Erode, India. She has published 5 research papers in various international journals. Area of interest includes low power VLSI design, VLSI signal processing and Network Security. She is the life member of ISTE and IETE.

SenthilKumar Athappan has received his B.E. Degree in Electrical and Electronics Engineering from PSG College of Technology in 1995 and M.E. Degree in VLSI Systems from Regional Engineering College (NIT), Trichy, in 2001. He completed Ph.D in the area of VLSI Signal Processing from Anna University, Chennai in 2009. Currently, he is the Professor and Head, Department of EIE, Dr. Mahalingam College of Engineering and Technology, Pollachi, India. He presented more than 30 papers in various national and international conferences. He published 13 papers in International Journals in the area of Low power VLSI design. His area of interest includes Embedded System, VLSI Signal Processing, and Industrial Automation.